Apple IIGS
#17: Application Memory Management and the MMStartUp User ID

Revised by: Steven Glass & Rich Williams November 1988
Written by: Jim Merritt June 1987

This Technical Note describes a technique which permits an application to dispose of any memory it has used with a single Memory Manager call without clobbering other system components or itself.

Ground Rules for Application Memory Usage

Apple IIGS programs must be responsible for allocating and disposing of any memory they use, over and above that which the operating system itself gives them. In general, no IIGS program should use any memory except that which the Memory Manager has explicitly granted to it. A program may request additional memory for its own use at any time with one or more calls to the `NewHandle` routine. At program termination, the application is responsible for explicitly disposing of any memory that it explicitly acquired, and if it fails to do so, it could leave the IIGS memory management system in a corrupted state.

You may dispose of memory on a handle-by-handle basis, or you may dispose of it *en masse* by calling `DisposeAll`, but you should **never** use `DisposeAll` with the user ID that the `MMStartUp` routine provides. This user ID is the "master user ID" for the application, and it tags the memory space which the operating system reserves for the program's code and static data at load time. Calling `DisposeAll` with this user ID results in immediate deallocation of the memory in which the calling program resides; therefore, an application which allocates dynamic data space using only the user ID that `MMStartUp` gives it should not use `DisposeAll` to deallocate that space, but rather use `DisposeHandle` to deallocate it handle by handle.

Cleaning Up With DisposeAll

It is possible, however, for a program to use a different, unique user ID when allocating its own RAM, then pass that user ID to `DisposeAll` when it terminates to deallocate all of its private memory at once without endangering itself or other parts of the IIGS system. With this technique, the question is how best to acquire a new user ID? One method to acquire a new user

ID is to request a completely new one of the appropriate type from the User ID Manager in the Miscellaneous Tools. In this case, when the application terminates, it must not only deallocate the memory it used, but also the additional user ID which it requested from the User ID Manager.

Actually, it is not necessary for a program to acquire a completely new user ID to use `DisposeAll` without clobbering itself. Instead, the application may modify the `auxID` field of the master user ID which `MMStartUp` assigns to create a unique user ID for allocating its own memory. The 16-bit user ID contains the `auxID` field in bits $8 – $B. The value of this field, which may range from $0 to $F, is always zero in the application's master user ID, but you can fill it with any non-zero value to create up to 15 new and distinct user IDs, each of which you can pass to `NewHandle` to allocate memory.and to `DisposeAll` to deallocate memory without endangering the memory tagged by the master user ID. The following assembly code fragment illustrates this technique:

```
        ; assumes full native mode
                pushword #0                 ; room for user ID
                _MMStartUp
                pla                         ; master user ID
                sta MasterID
                ora  #$0100                 ; auxID:= 1

        ;       (COULD HAVE BEEN ANYTHING FROM $1 to $F)

                sta MyID                    ; use this to allocate private memory
                ...
                etc.
                ...

        ; ready to exit program
                pushword MyID
                _DisposeAll                 ; dumps only my own RAM

; now do any remaining processing related to termination
```

You do not need to explicitly deallocate any user ID that you derive by changing the `auxID` field of a valid master user ID. When the system (usually the one to deallocate the master) deallocates the master user ID, it also deallocates its derivatives.

One Word of Caution

Several of the Memory Manager's "All" calls (e.g., `DisposeAll`) treat a zeroed `auxID` field as a wildcard which matches any value that the field may contain, thus if you call `DisposeAll` with the application's master user ID (where the `auxID` field is zero), the Memory Manager will not only deallocate all memory belonging to the master user ID, but also all handles and memory segments that are associated with user IDs which are derived from that master. The Loader's `UserShutDown` mechanism typically executes such a call when cleaning up after a normal (i.e., non-restartable) application to keep the memory management system from clogging. This action is purely a defensive measure, and well-behaved applications – particularly restartable ones – should dispose of their own memory and never rely upon the operating system to clean up after them.

Further Reference

- *Apple IIGS Toolbox Reference*, Volume 1